Materialized Views stellen ein wichtiges Konzept im Rahmen der Data-Warehouse-Technologien dar. In Data Warehouses dienen sie in erster Linie der Steigerung der Abfrage-Performance durch die Vorberechnung von Abfrageergebnissen.

Aktualisierungsoptionen bei Materialized Views

Timo Bergenthal, OPITZ CONSULTING Essen GmbH

Materialized Views können wie eine Tabelle performant abgefragt und wiederverwendet werden und mit einfachen Datenbankmitteln auf aktuellem Stand gehalten werden. In Umgebungen mit verteilten Datenbanken können diese Views Tabellen replizieren und so dafür sorgen, dass die Tabelleninhalte stets in mehreren Datenbanken lokal zugreifbar sind. Dieser Artikel beschreibt den grundsätzlichen Aufbau, die Möglichkeiten, die sie im Zuge einer Aktualisierung bieten, und einige Aspekte, die in diesem Zusammenhang beachtet werden sollten.

Materialized Views kombinieren die Speicherung von Daten mit der Speicherung der zugehörigen Abfrage. Sie unterscheiden sich deutlich von einer klassischen View:

- Sie belegen Speicher
- Sie können wie eine Tabelle abgefragt werden, ohne dass die zur Materialized View gehörige Abfrage erneut ausgeführt werden muss
- Sie müssen zu bestimmten Zeitpunkten aktualisiert werden, damit Abfrage und Inhalte einander entsprechen

Diese Kombination spiegelt sich in den Datenbank-Objekten wider, die im Zusammenhang mit einer Materialized View stehen. So werden bei der Erstellung einer Materialized View üblicherweise drei Datenbank-Objekte erzeugt, die sich in der Datenbank-View „DBA_OBJECTS“ wiederfinden (siehe Tabelle 1).

Die mit der Materialized View gespeicherten Meta-Informationen umfassen:

- Die SQL-Abfrage, auf der die Daten basieren
- Angaben zu Art und Zeitpunkt der Datenaktualisierung
- Die Angabe, ob Abfragen auf eine oder mehrere Quelltabellen der Materialized View dynamisch und transparent so umgeschrieben werden sollen, dass der Zugriff auf diese Quelltabellen ersetzt wird durch den Abruf der Materialized View. Durch dieses Feature – „Query Rewrite“ genannt – lassen sich andere Abfragen unter Umständen drastisch beschleunigen, ohne die Abfrage oder eine der Quelltabellen in irgendeiner Weise zu modifizieren
- Die Angabe, ob die Daten in der Tabelle der Materialized View manuell geändert werden dürfen; diese Änderungen können unter Umständen in die Quelltabelle weiterpropagiert werden

Das besondere Augenmerk in diesem Artikel liegt auf der Aktualisierung einer Materialized View. Durch Änderungen in Tabellen, die innerhalb der Abfrage referenziert werden – den sogenannten „Master-Tabellen“ –, muss üblicherweise auch die Materialized View auf den neuesten Stand gebracht werden. Dies gewährleistet, dass Abfrage und Daten einander konsistent gegenüberstehen.

Die Aktualisierung einer Materialized View kann durch drei unterschiedliche Ereignisse ausgelöst werden. Die Entscheidung, welches dieser Ereignisse zur Aktualisierung führt, muss bereits im Zuge der Erstellung der Materialized View getroffen werden.

Standard bei der Erstellung einer Materialized View ist die Option „ON DEMAND“. Diese besagt, dass es keinen Automatismus gibt, der zur Aktualisierung der Materialized View führt. Die Aktualisierung wird nur durch Aufruf einer entsprechenden Operation initiiert, der Prozedur „REFRESH“ im Package „DBMS_MVIEW“. Die genannte Prozedur kennt mehrere Übergabeparameter, von denen einige von Bedeutung sind (siehe Tabelle 2).

Auf die möglichen Aktualisierungsmethoden sowie eine weitere, wichtige Auswirkung des Parameters „ATOMIC_REFRESH“ wird im Absatz „Complete Refresh“ eingegangen.

Beim Anlegen der Materialized View legt die Klausel „START WITH“ fest, wann die erste Aktualisierung erfolgen soll. Die „NEXT“-Anweisung gibt vor, in welchem zeitlichen Intervall eine Aktualisierung erfolgen soll. An beide Klauseln schließt sich ein Ausdruck an, der zu einem Wert vom Datentyp „DATE“ ausgewertet werden kann. Nachfolgend wird eine Materialized View erstellt, die erstmals am 1. De-

Typ	Zweck
TABLE	Speicherung der Daten
INDEX	Performancesteigerung bei der Aktualisierung der Materialized View
MATERIALIZED VIEW	Speicherung von Metainformationen; Objekt trägt den gleichen Namen wie die Tabelle

Tabelle 1

	Name	Bedeutung	Standard in der DB-Version 11.1.0.6
1	list / tab	kommaseparierte Liste oder eine PL/SQL-Tabelle der Materialized Views, die aktualisiert werden sollen	
2	method	Methode, mit der die Materialized View aktualisiert werden soll. Diese Angabe ist für die Aktualisierung bindend und wird unabhängig von der Definition der Materialized View verwendet.	NULL, d.h. Definition der Materialized View kommt zum Tragen
9	atomic_ refresh	Angabe, ob die Aktualisierung der Materialized Views innerhalb einer einzigen Transaktion stattfinden soll: TRUE für Aktualisierung in einer Transaktion, FALSE sonst. Bei Aktualisierung in einer Transaktion kann diese im Falle eines Fehlers wieder komplett zurückgerollt werden.	TRUE

Tabelle 2

zember 2010 um 0 Uhr und ab diesem Moment täglich um 12 Uhr mittags aktualisiert wird:

```
CREATE MATERIALIZED VIEW mat-
view REFRESH FAST
START WITH TO_DATE(‚20101201‘,
‚YYYYMMDD‘)
NEXT TRUNC(SYSDATE + 1) + 1/2
WITH PRIMARY KEY AS SELECT …
```

Wird die Materialized View mittels der Klausel „ON COMMIT“ angelegt, so ist eine Diskrepanz zwischen Abfrage und Daten per Definition ausgeschlossen. Auf jede Änderung, die an den Master-Tabellen durchgeführt und durch COMMIT bestätigt wird, folgt eine Aktualisierung der Daten der Materialized View. Diese Aktualisierung geschieht innerhalb des „COMMIT“-Prozesses, sodass bei größeren Transaktionen die Ausführung des „COMMIT“-Befehls länger dauern kann als gewohnt. Dies kann sich negativ auf einen vorgelagerten Bewirtschaftungsprozess auswirken und insbesondere bei vielen simultanen Transaktionen kritisch sein. Daher ist diese Option, insbesondere in Systemen mit vielen parallel arbeitenden Anwendern, nur mit Bedacht einzusetzen. Bei referenzierten Tabellen auf Remote-Datenbanken oder Objekttyp-Spalten in der Materialized View ist diese Option grundsätzlich nicht einsetzbar.

Die Aktualisierung der Daten kann entweder komplett oder inkrementell erfolgen, wobei Oracle die etwas trügerischen Bezeichnungen „COMPLETE“ beziehungsweise „FAST REFRESH“ etabliert hat. Beide Methoden unterscheiden sich grundlegend voneinander und werden im Folgenden detailliert erläutert. Der sogenannte „PCT-Refresh“ folgt danach.

Der „Complete Refresh“ einer Materialized View ist immer möglich und nicht an bestimmte Voraussetzungen geknüpft. Dabei werden sämtliche Daten der Materialized View mithilfe der hinterlegten Abfrage neu berechnet. Der Parameter „ATOMIC_REFRESH“ im Aufruf der Prozedur „DBMS_MVIEW. REFRESH“(siehe „Aktualisierung auf Anfrage“) spielt insbesondere in diesem Szenario eine wichtige Rolle. Die Beschränkung auf nur eine Transaktion unterbindet das performante Leeren der Tabelle per TRUNCATE und die Verwendung des direkten Pfades bei der anschließenden Neuberechnung der Materialized View. Eine Prüfung, ob der genannte Parameter auf „FALSE“ gesetzt werden kann, ist folglich als obligatorisch zu betrachten. Abbildung 1 zeigt den Ablauf eines „Complete Refresh“.

Will man eine Materialized View standardmäßig per „Complete Refresh“ aktualisieren, muss die Klausel „REFRESH COMPLETE“ bei der Erstellung der Materialized View verwendet werden. Wird eine Materialized View damit aktualisiert, sind alle darauf aufbauenden Materialized Views ebenfalls per „Complete Refresh“ zu erneuern.

Der Begriff „Fast Refresh“ ist etwas missverständlich, da diese Methode unter Umständen mehr Zeit in Anspruch nehmen kann als ein „Complete Refresh“. Dies ist dann der Fall, wenn ein großer Anteil der Daten in den Mastertabellen geändert wurde. Sinnvoller wäre daher ein Ausdruck wie „Inkremental Refresh“. Im Zuge des „Fast Refresh“ werden sämtliche Änderungen an den Master-Tabellen, die sich seit der letzten Aktualisierung ereignet haben, in die Daten der Materialized View eingepflegt.

Der „Fast Refresh“ ist im Gegensatz zum „Complete Refresh“ nicht immer möglich, sondern unterliegt bestimmten Voraussetzungen. So lassen sich beispielsweise Materialized Views

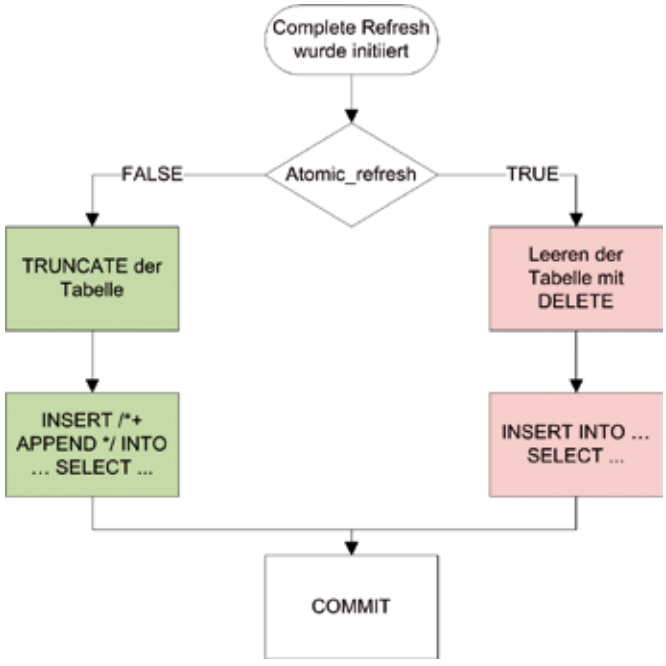

Abbildung1: Ablauf des Complete Refresh

mit analytischen Funktionen nicht per „Fast Refresh“ aktualisieren, da die Ergebnisse analytischer Funktionen meist nicht nur auf Basis eines Deltas neu berechnet werden können.

Es gibt eine weitere Methode, mit der die Datenbank die Aktualisierung einer Materialized View – geknüpft an einige Voraussetzungen – durchführen kann. Diese baut auf dem sogenannten „Partition Change Tracking (PCT)“ auf. PCT bedeutet, dass die Datenbank in der Lage ist, Zeilen einer Materialized View exakt einer Partition einer partitionierten Master-Tabelle zuzuordnen. Dazu gehört auch, dass bei Änderungen an einer partitionierten Master-Tabelle nicht direkt die ganze Materialized View in den Status „STALE“ („nicht auf aktuellem Stand“, im Gegensatz zu „FRESH“) wechselt, sondern gegebenenfalls nur die Zeilen der Materialized View, die der geänderten Partition einer Mastertabelle entstammen. Diese Eigenschaft macht sich Query Rewrite zunutze, sodass aktuelle Teile der Materialized View weiter uneingeschränkt für Abfragen genutzt werden können.

Des Weiteren kann PCT bei der Aktualisierung einer Materialized View genutzt werden. Kommt der „PCT-Refresh“ zum Einsatz, werden alle Daten in der Materialized View, die den geänderten Partitionen der Mastertabelle entstammen, gelöscht und anschließend komplett neu berechnet. Das Vorgehen ist so vergleichbar mit dem „Complete Refresh“, allerdings beschränkt auf die geänderten Partitionen. Partition-Maintainance-Operationen wie „TRUNCATE/DROP PARTITION“ werden dabei erkannt und bei der Aktualisierung berücksichtigt, sodass ein anschließender „Complete Refresh“ überflüssig ist.

Werden Materialized Views nicht nur zu Replikationszwecken verwendet, kommen üblicherweise Joins oder Aggregationen in der Abfrage der Materialized View zum Einsatz. Die Datenbank-View „DBA_REGISTERED_MVIEWS“ gibt als Aktualisierungsmethode in diesem Fall „ROWID“ an, auch wenn sich die Aktualisierung unter Umständen gar nicht der ROWID bedient. So wird bei Materialized Views, in denen nach einer oder mehreren Spalten gruppiert wird, die Aktualisierung üblicherweise über ein oder mehrere „MERGE“-Statements durchgeführt. In der „ON“-Klausel sind dann alle Spalten enthalten, die für die Eindeutigkeit eines Datensatzes in der Materialized View sorgen. Per Definition sind dies gerade die Spalten, nach denen gruppiert wird. Die Aktualisierung kann in einem solchen Fall häufig durch die Verwendung eines Index zügiger vonstatten gehen. Dies erklärt, warum per Default bei der Erstellung einer Materialized View im gleichen Zuge ein Index auf der Tabelle angelegt wird. Die Option „USING NO INDEX“ unterdrückt diesen.

Ein weiteres Beispiel für Materialized Views im Data-Warehouse-Umfeld bilden Materialized Views mit Joins ohne Aggregation. In diesem Fall findet die Aktualisierung auf Basis der ROWID statt. Es muss jedoch die ROWID aller beteiligten Tabellen in der Materialized View enthalten sein. Dafür muss der Ersteller der Materialized View selbst Sorge tragen.

Die Information, welche Änderungen die Master-Tabellen erfahren haben, steht der Datenbank nicht ohne Weiteres zur Verfügung. Zu diesem Zweck muss für jede Master-Tabelle ein sogenannter „Materialized View Log“ angelegt werden, der per internem Datenbank-Trigger Änderungen in der zugehörigen Master-Tabelle protokolliert. Ohne dessen Existenz kann keine Materialized View erstellt werden, die per „Fast Refresh“ aktualisierbar ist.

Ein Materialized View Log entspricht einer Tabelle, die üblicherweise den Namen „MLOG$_mastertabelle“ trägt, im gleichen Schema liegt wie die zugehörige Mastertabelle und standardmäßig einige Spalten mit technischen Informationen beinhaltet. Weitere Spalten können je nach Bedarf hinzukommen. Das Einfügen zusätzlicher Spalten wird gesteuert durch die Schlüsselwörter „ROWID“, „PRIMARY KEY“ und/oder „SEQUENCE“ beziehungsweise durch die explizite Angabe weiterer Spalten der Master-Tabelle. Von der Auswahl dieser Spalten hängt ab, welche Spalten in der Abfrage der Materialized View referenziert werden.

In den meisten Fällen muss die ROWID in den Materialized View Log integriert sein. Dies gilt insbesondere für Materialized Views, die nur Joins und keine Aggregationen enthalten. Zusätzlich muss bei Materialized Views, in denen eine Aggregation durchgeführt wird, jede referenzierte Spalte auch im Materialized View Log enthalten sein. Des Weiteren sollte in diesem Fall die „SEQUENCE“ mit im Log enthalten sein, wenn unterschiedliche DML-Operationen auf der Mastertabelle abgesetzt werden. Zur Hilfestellung bei der Erstellung der Materialized View Logs stellt die Datenbank übrigens die Prozedur „DBMS_MVIEW.EXPLAIN_MVIEW“ zur Verfügung.

Fazit

Materialized Views dienen der Performance-Steigerung in Data-Warehouse-Systemen. Damit sie dieser Aufgabe gerecht werden, sind sie mit Bedacht auszuwählen. Im Idealfall sollte eine große Anzahl von zeitkritischen oder immer wieder ausgeführten Abfragen von der Materialized View profitieren, sodass die Vorteile klar dominieren gegenüber den Nachteilen des erhöhten Speicherbedarfs und des zusätzlichen Rechenaufwands für die Aktualisierung der Materialized View. Ein weiterer Vorteil von Materialized Views ist die vereinfachte Aktualisierung der Inhalte, die durch einen simplen Prozeduraufruf durchgeführt werden kann, die den Möglichkeiten einer Tabelle weit überlegen ist.

Timo Bergenthal
OPITZ CONSULTING GmbH
timo.bergenthal@opitz-consulting.com